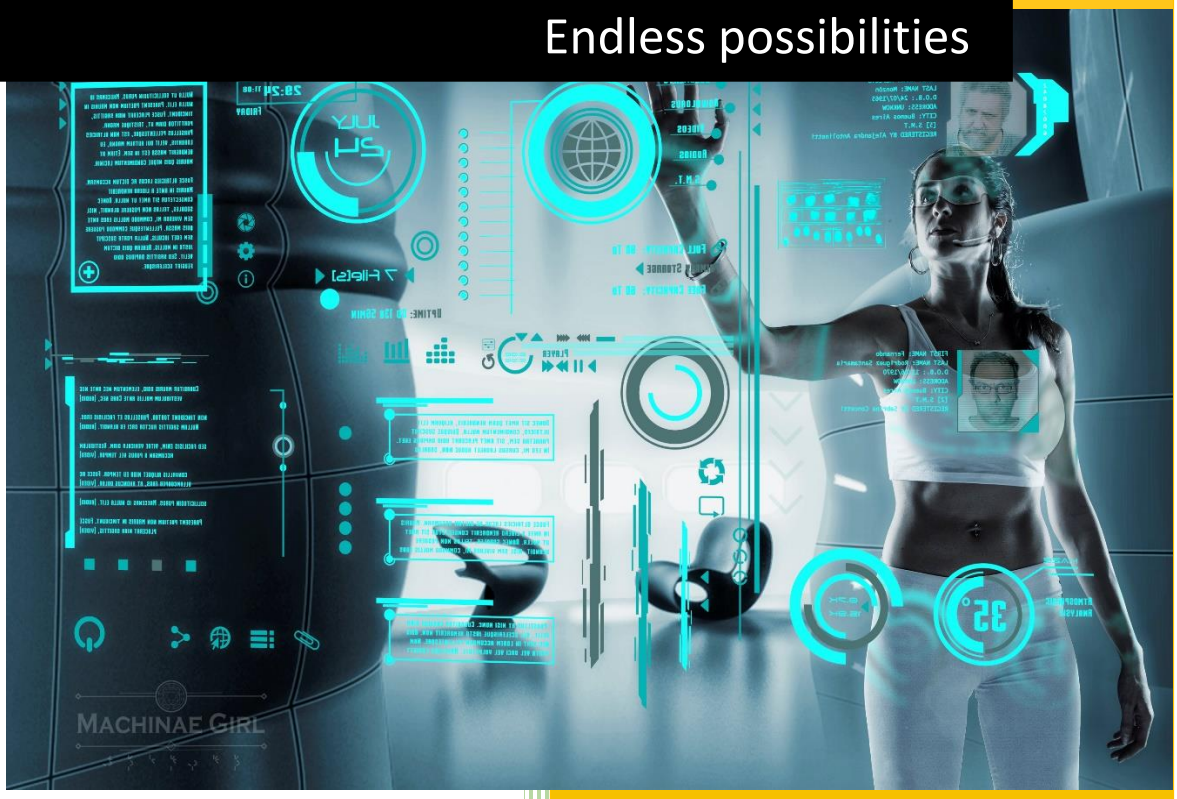
2017

SQL – Databases

Endless possibilities



Denys Lopaiev

denys.lopaiev@gmail.com

11/15/2017

Contents

Introduction

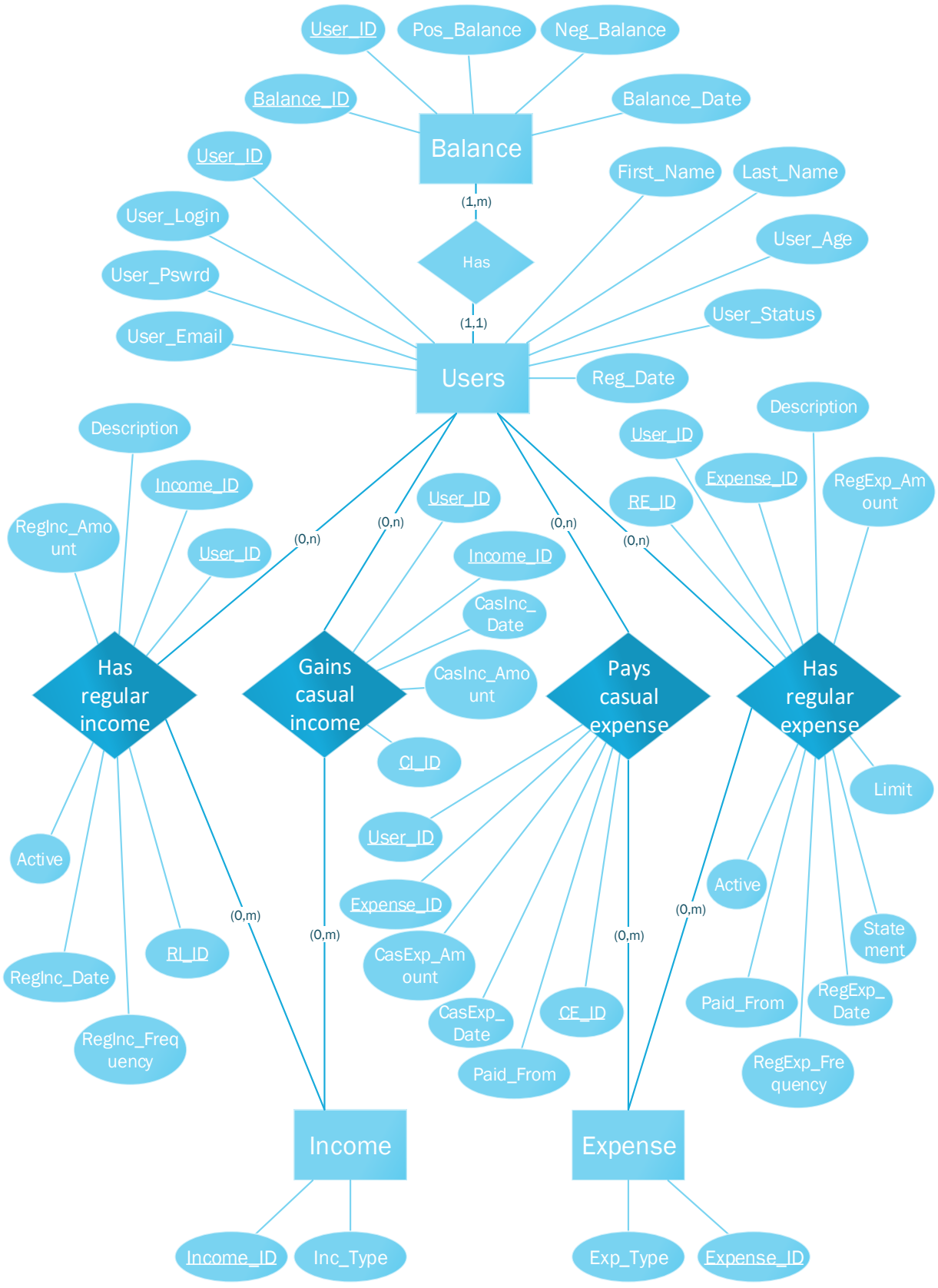
With some exceptions, all of us have dreams and goals. And achieving them it's a big part of our happiness. Some people want to have good education to get better positions in the future and build their prestige career, some of them thinking about buying their first house or car, someone wants to travel. Lots of people who want to start their own business or invest money and make profit. And modern life has not lost yet those who want to help other people.

Everything written above are some examples from thousands, maybe millions different desires. Nowadays it's getting easier to achieve everything you want using various tools and propositions. I would say it's just matter of hardworking and your reputation. And all these tools become available to you. But at the same time it's getting easier to lose common sense when all limits are out. And this is where you can dive into the world of big debts.

Having this picture in your mind, let's think about two major cash flows: income and expense. Nothing special and very simple idea behind: money come money go. Pretend, income and expense are two weights that we place on the scales. Are they equal? I assume you caught yourself on this thought at least once: "Do I spend more than I earn?". Here is the place where we should make an addition to our simple idea: money come -> balance -> money go. How many of you think about balancing your cash flows? It’s not enough just to think about it though. We need actions. We need to take control over our cash flows.

Considering this, the main idea of current project was to develop database for the future personal financial management system that would help to keep track of all the personal incomes and expenses, providing easy tools for better financial analysis.

Entity Relationship Diagram



The Entity Relationship Diagram (ERD) Illustrated above reflects relationships between user, his balance, income and expense. Consider following business rules:

- User can have as many incomes as possible, as well as no income at all. Income divided into 2 groups: regular and casual.
- User can have as many expenses as he wants, as well as no expense at all. Expense divided into 2 groups: regular and casual.
- For regular expense user can establish limit. In this case he won't be able to spend over the established amount until it's partially or completely paid out.
- User can have positive and negative balance updated daily.
- Casual and regular expense could be paid directly from positive user's balance or can make up another regular expense. User has to determine from where these expenses are paid.

Current ERD is scalable and might expand over time.

Table creation with DDL

Based on represented ERD we can create 8 tables within database utilizing Microsoft SQL Server Management Studio and required DDL statements.

```
CREATE TABLE Users (
User_ID int IDENTITY(1,1) NOT NULL,
User_Login varchar(50) NOT NULL,
User_Pswrd varchar(50) NOT NULL,
User_Email varchar(100) NOT NULL,
First_Name varchar(255),
Last_Name varchar(255),
User_Age int,
User_Status varchar(255),
Reg_Date datetime NOT NULL,
CONSTRAINT PK_User PRIMARY KEY (User_ID)
);
```

```
CREATE TABLE Balance (
Balance_ID int IDENTITY(1,1) NOT NULL,
User_ID int NOT NULL,
Pos_Balance money NOT NULL,
Neg_Balance money NOT NULL,
Balance_Date datetime NOT NULL,
CONSTRAINT PK_Balance PRIMARY KEY
(Balance_ID),
CONSTRAINT FK_UserHasBalance FOREIGN KEY
(User_ID)
REFERENCES Users(User_ID)
ON UPDATE NO ACTION ON DELETE CASCADE
);
```

```
CREATE TABLE Income (
Income_ID int IDENTITY(1,1) NOT NULL,
Inc_Type varchar(255) NOT NULL,
CONSTRAINT PK_Income PRIMARY KEY
(Income_ID)
);
```

```
CREATE TABLE Expense (
Expense_ID int IDENTITY(1,1) NOT NULL,
Exp_Type varchar(255) NOT NULL,
CONSTRAINT PK_Expense PRIMARY KEY
(Expense_ID)
);
```

```
CREATE TABLE RegularIncome (
RI_ID int IDENTITY(1,1) NOT NULL,
User_ID int NOT NULL,
Income_ID int,
Description varchar(255),
RegInc_Amount money NOT NULL,
RegInc_Date date NOT NULL,
RegInc_Frequency smallint NOT NULL,
Active bit NOT NULL,
CONSTRAINT PK_RegularIncome PRIMARY KEY
(RI_ID),
CONSTRAINT FK_UserHasRegularIncome FOREIGN
KEY (User_ID)
REFERENCES Users(User_ID)
ON UPDATE NO ACTION ON DELETE CASCADE,
CONSTRAINT FK_RegularIncomeType FOREIGN
KEY (Income_ID)
REFERENCES Income(Income_ID)
ON UPDATE SET NULL ON DELETE SET NULL
);
```

```
CREATE TABLE RegularExpense (
RE_ID int IDENTITY(1,1) NOT NULL,
User_ID int NOT NULL,
Expense_ID int,
Description varchar(255),
Limit money,
RegExp_Amount money NOT NULL,
Statement money,
RegExp_Date datetime NOT NULL,
RegExp_Frequency smallint NOT NULL,
Paid_From varchar(255),
Active bit NOT NULL,
CONSTRAINT PK_RegularExpense PRIMARY KEY
(RE_ID),
CONSTRAINT FK_UserHasRegularExpense
FOREIGN KEY (User_ID)
REFERENCES Users(User_ID)
ON UPDATE NO ACTION ON DELETE CASCADE,
CONSTRAINT FK_RegularExpenseType FOREIGN
KEY (Expense_ID)
REFERENCES Expense(Expense_ID)
ON UPDATE SET NULL ON DELETE SET NULL
);
```

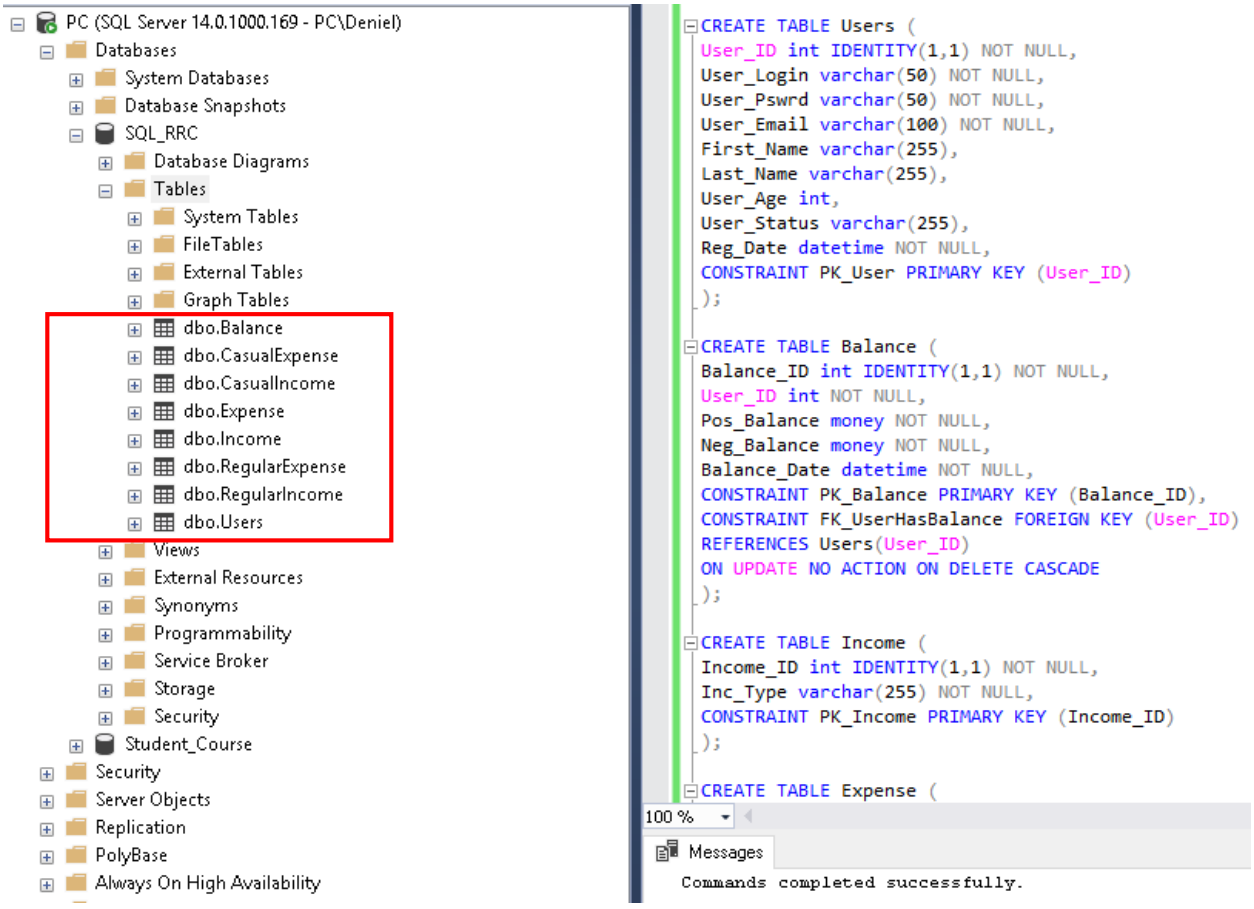
```
CREATE TABLE CasualIncome (
CI_ID int IDENTITY(1,1) NOT NULL,
User_ID int NOT NULL,
Income_ID int,
CasInc_Amount money NOT NULL,
CasInc_Date datetime NOT NULL,
```

```
CREATE TABLE CasualExpense (
CE_ID int IDENTITY(1,1) NOT NULL,
User_ID int NOT NULL,
Expense_ID int,
CasExp_Amount money NOT NULL,
CasExp_Date datetime NOT NULL,
```

```
CONSTRAINT PK_CasualIncome PRIMARY KEY
(CI_ID),
CONSTRAINT FK_UserGainsCasualIncome
FOREIGN KEY (User_ID)
REFERENCES Users(User_ID)
ON UPDATE NO ACTION ON DELETE CASCADE,
CONSTRAINT FK_CasualIncomeType FOREIGN KEY
(Income_ID)
REFERENCES Income(Income_ID)
ON UPDATE SET NULL ON DELETE SET NULL
);
```

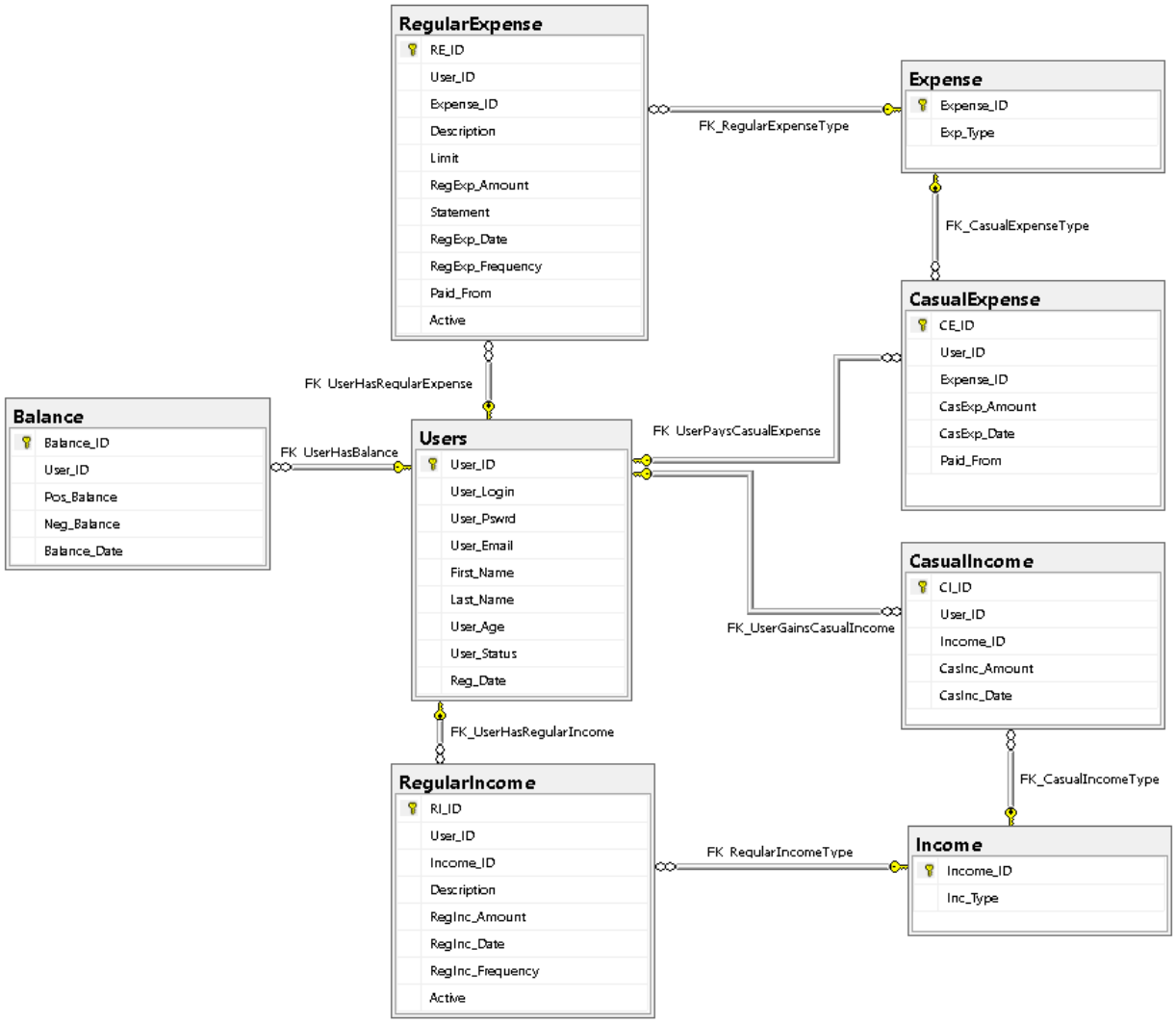
```
Paid_From varchar(255),
CONSTRAINT PK_CasualExpense PRIMARY KEY
(CE_ID),
CONSTRAINT FK_UserPaysCasualExpense
FOREIGN KEY (User_ID)
REFERENCES Users(User_ID)
ON UPDATE NO ACTION ON DELETE CASCADE,
CONSTRAINT FK_CasualExpenseType FOREIGN
KEY (Expense_ID)
REFERENCES Expense(Expense_ID)
ON UPDATE SET NULL ON DELETE SET NULL
);
```

As a result, 8 tables were created.



Database Diagram

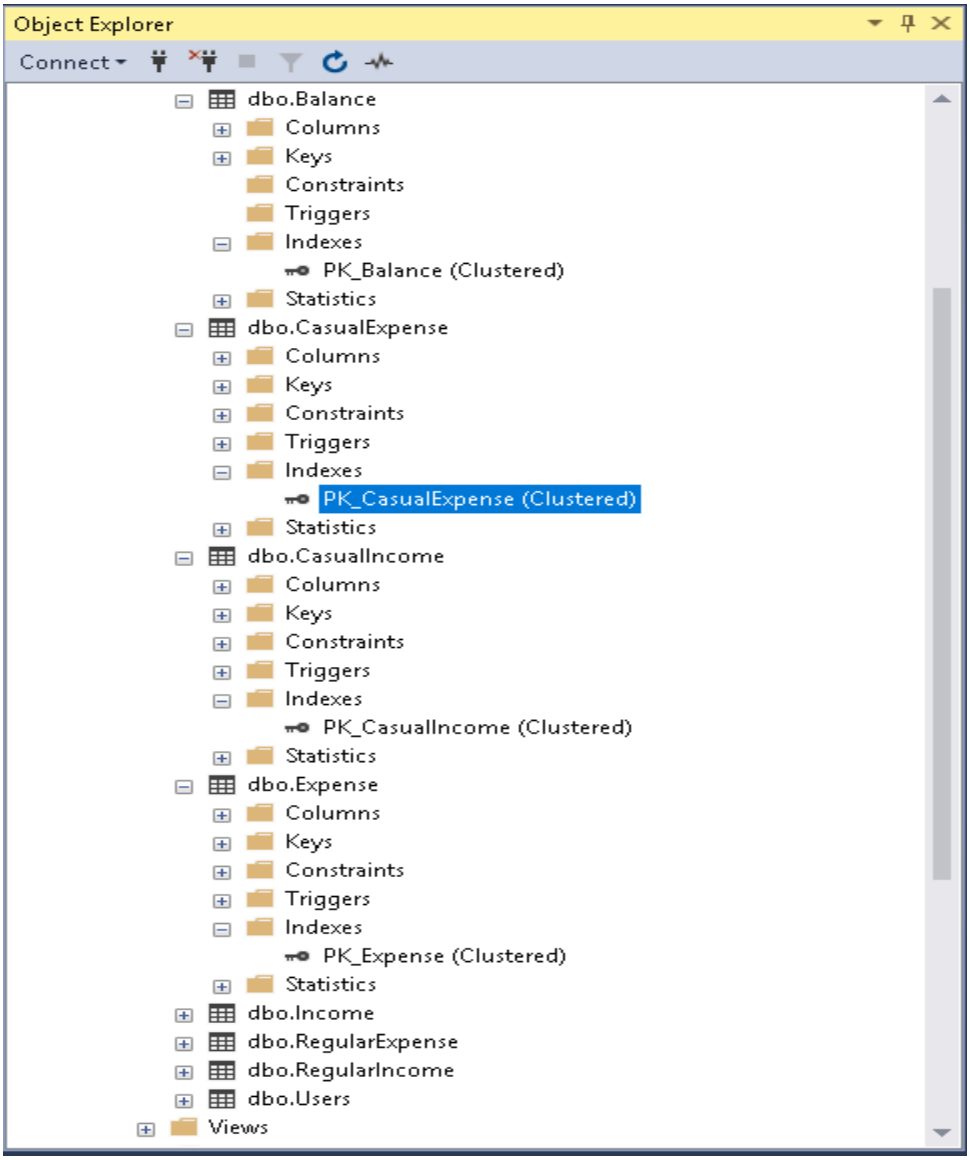
Database diagram below shows the tables contained in the database and relationships that exist between them.



Performance tuning with indices

Indices are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book. Table can contain clustered and non-clustered indices.

During table creation process clustered indices were auto-created for each table. Following Microsoft documentation, there can be only one clustered index per table, because the data rows themselves can be sorted in only one order.



Although indexes are intended to enhance a database's performance, there are times when they should be avoided.

The following guidelines indicate when the use of an index should be reconsidered:

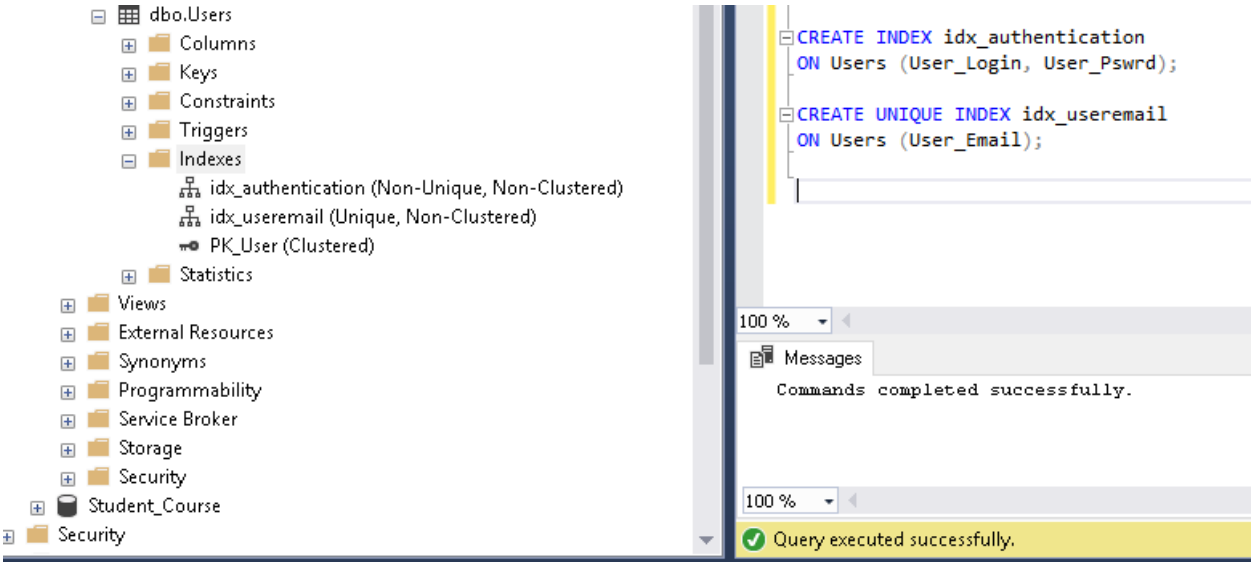
- Indexes should not be used on small tables.
- Tables that have frequent, large batch updates or insert operations.
- Indexes should not be used on columns that contain a high number of NULL values.
- Columns that are frequently manipulated should not be indexed.

Income and Expense tables belong to small tables, Balance, RegularExpense, CasualExpense, RegularIncome and CasualIncome tables will be frequently updated because all of them will contain money amounts and dated records of all user operations with them.

For table Users would make more sense to use indices – updates or insert operations are not very frequent in this case.

First non-clustered index will be created for columns User_Login and User_Pswrd, second non-clustered index for column User_Email. These columns will be frequently searched against, e.g. user authentication, password recovering actions.

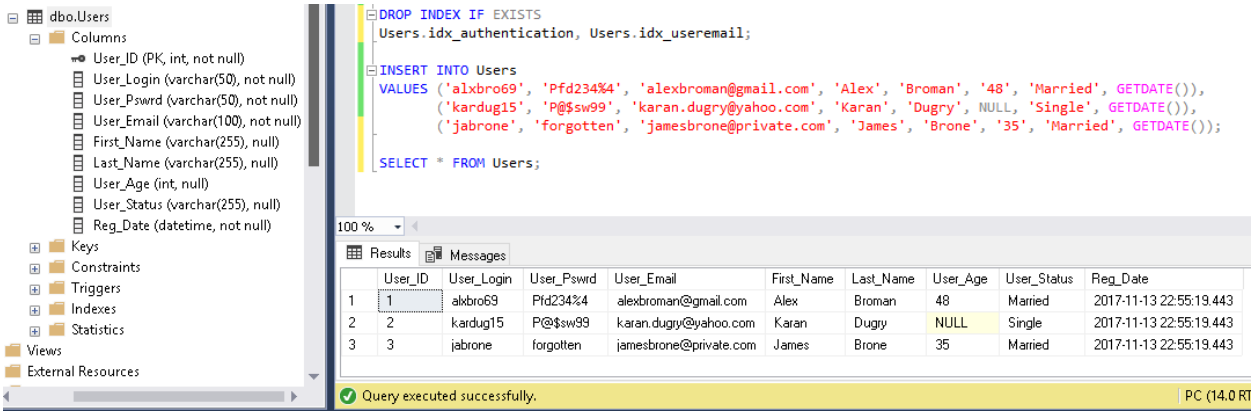
```
CREATE INDEX idx_authentication
ON Users (User_Login, User_Pswrd);

CREATE UNIQUE INDEX idx_useremail
ON Users (User_Email);
```



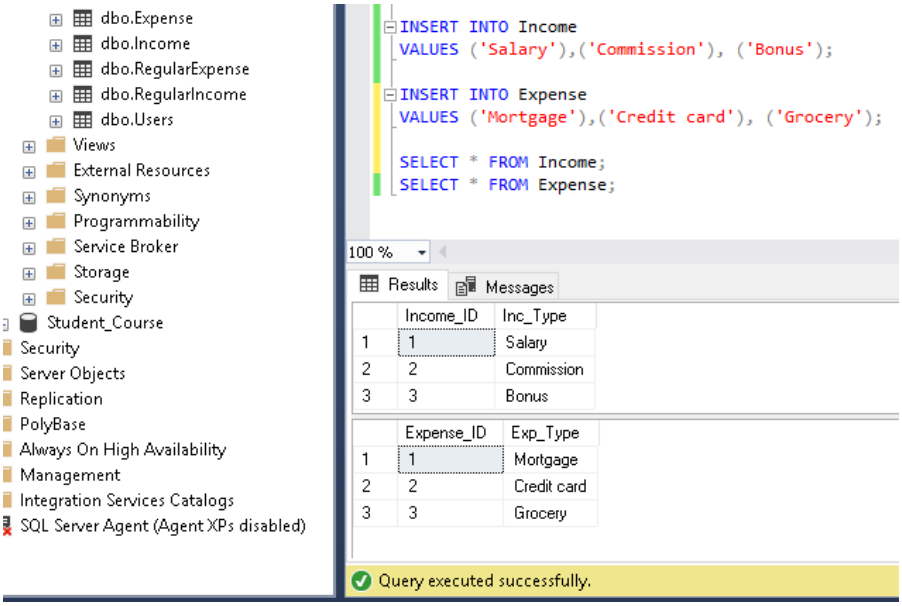
Use of DML

Three rows of data will be added to each table using insert statements.

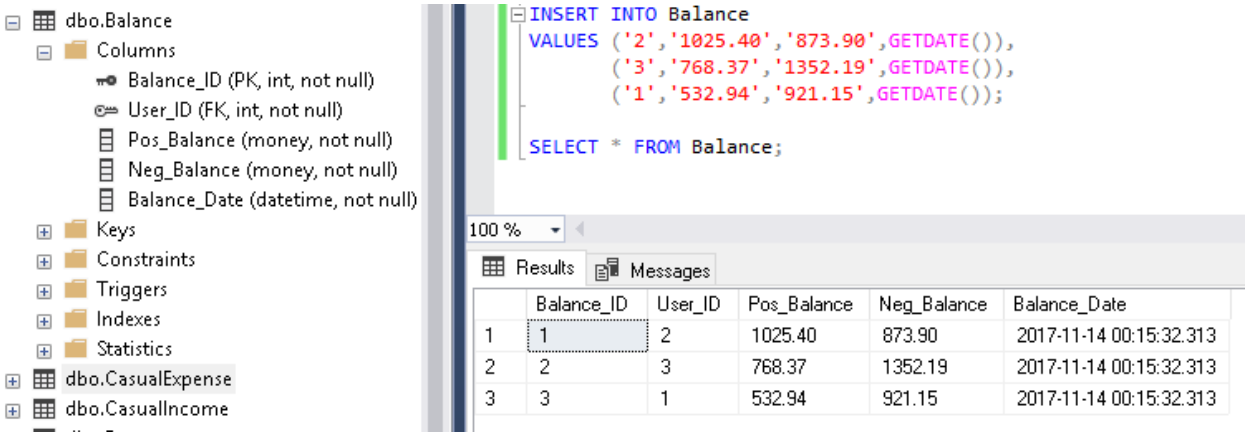
```
INSERT INTO Users
VALUES ('alxbro69', 'Pfd234%4', 'alexbroman@gmail.com', 'Alex', 'Broman', '48',
'Married', GETDATE()),
         ('kardug15', 'P@$sw99', 'karan.dugry@yahoo.com', 'Karan', 'Dugry', NULL,
'Single', GETDATE()),
       ('jabrone', 'forgotten', 'jamesbrone@private.com', 'James', 'Brone', '35',
'Married', GETDATE());

SELECT * FROM Users;
```




```
INSERT INTO Income
VALUES ('Salary'),
       ('Commission'),
       ('Bonus');

INSERT INTO Expense
VALUES ('Mortgage'),
       ('Credit card'),
       ('Grocery');

SELECT * FROM Income;
SELECT * FROM Expense;
```

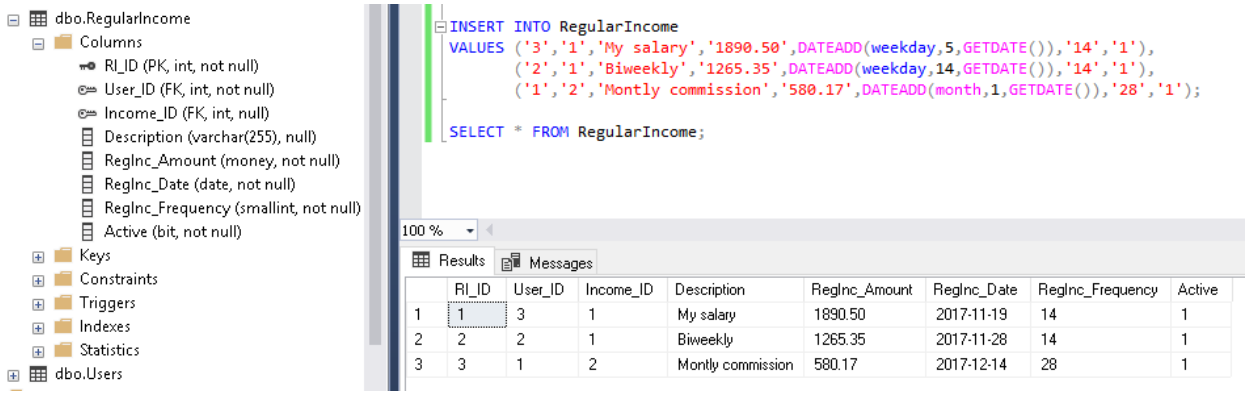


```
INSERT INTO Balance
VALUES ('2','1025.40','873.90',GETDATE()),
       ('3','768.37','1352.19',GETDATE()),
        ('1','532.94','921.15',GETDATE());

SELECT * FROM Balance;
```



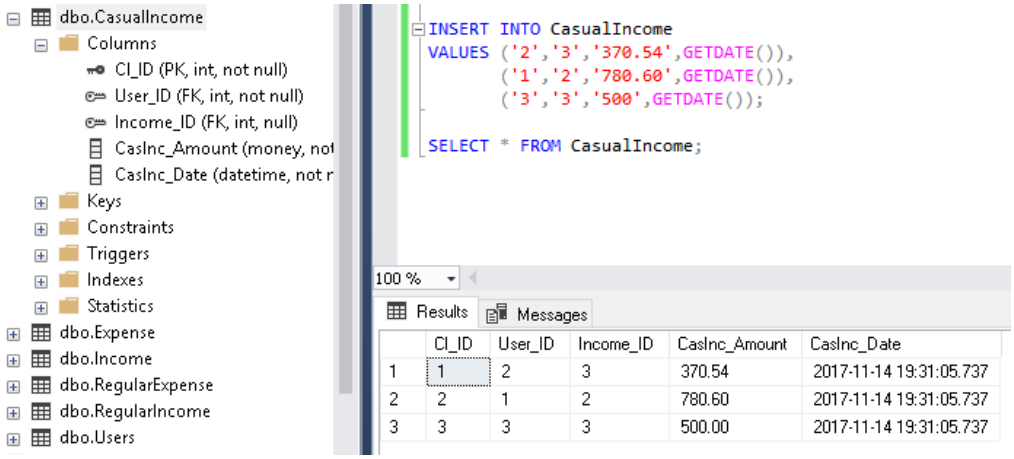
	Balance_ID	User_ID	Pos_Balance	Neg_Balance	Balance_Date
1	1	2	1025.40	873.90	2017-11-14 00:15:32.313
2	2	3	768.37	1352.19	2017-11-14 00:15:32.313
3	3	1	532.94	921.15	2017-11-14 00:15:32.313

```
INSERT INTO RegularIncome
VALUES ('3','1','My salary','1890.50',DATEADD(weekday,5,GETDATE()),'14','1'),
       ('2','1','Biweekly','1265.35',DATEADD(weekday,14,GETDATE()),'14','1'),
       ('1','2','Montly commission','580.17',DATEADD(month,1,GETDATE()),'28','1');

SELECT * FROM RegularIncome;
```

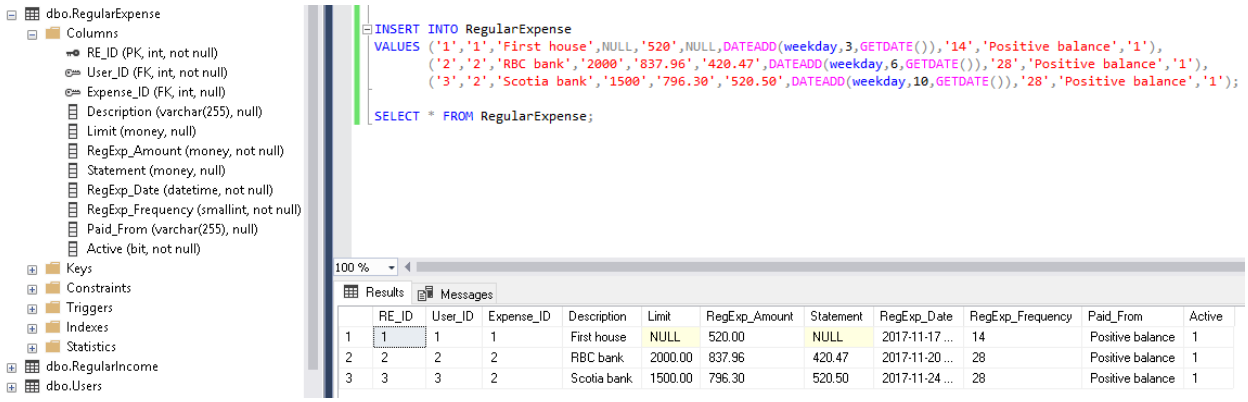


	RI_ID	User_ID	Income_ID	Description	RegInc_Amount	RegInc_Date	RegInc_Frequency	Active
1	1	3	1	My salary	1890.50	2017-11-19	14	1
2	2	2	1	Biweekly	1265.35	2017-11-28	14	1
3	3	1	2	Montly commission	580.17	2017-12-14	28	1

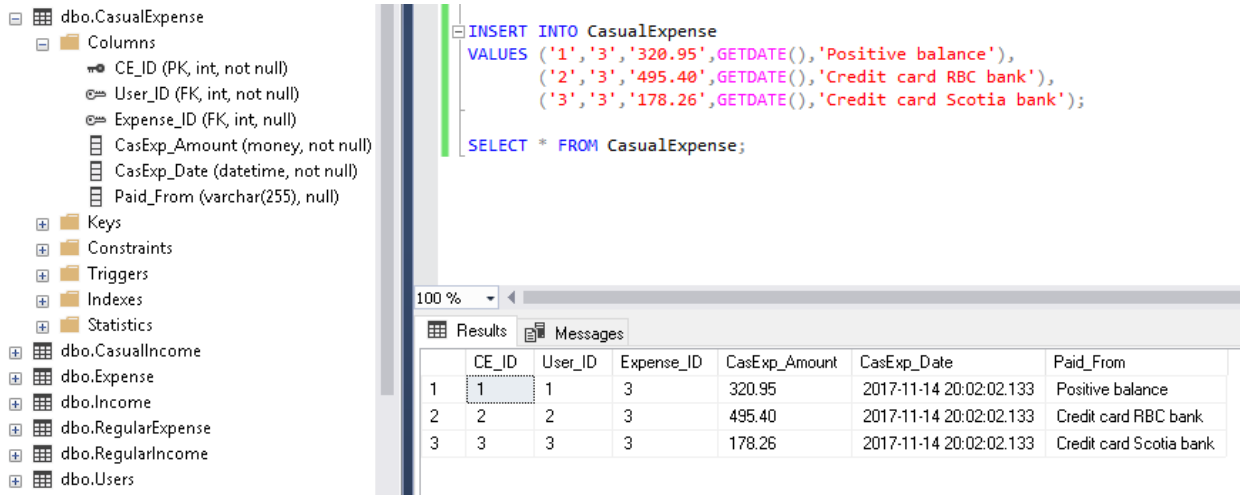
```
INSERT INTO CasualIncome
VALUES ('2','3','370.54',GETDATE()),
       ('1','2','780.60',GETDATE()),
       ('3','3','500',GETDATE());

SELECT * FROM CasualIncome;
```




```
INSERT INTO RegularExpense
VALUES ('1','1','First house',NULL,'520',NULL,DATEADD(weekday,3,GETDATE()),'14','Positive
balance','1'),
       ('2','2','RBC
bank','2000','837.96','420.47',DATEADD(weekday,6,GETDATE()),'28','Positive balance','1'),
       ('3','2','Scotia
bank','1500','796.30','520.50',DATEADD(weekday,10,GETDATE()),'28','Positive
balance','1');

SELECT * FROM RegularExpense;
```




```
INSERT INTO CasualExpense
VALUES ('1','3','320.95',GETDATE(),'Positive balance'),
       ('2','3','495.40',GETDATE(),'Credit card RBC bank'),
       ('3','3','178.26',GETDATE(),'Credit card Scotia bank');

SELECT * FROM CasualExpense;
```



Update and delete statements will be used to modify the data in the tables. By placing these DML statements into a transaction we will be able to commit or rollback all the changes.

```
BEGIN TRANSACTION;
UPDATE Users
SET User_Status = 'Single'
WHERE User_Login = 'jabrone';

UPDATE Balance
SET Pos_Balance = '1390.20', Balance_Date = GETDATE()
WHERE User_ID = '3';

UPDATE RegularIncome
SET RegInc_Amount = '450.00'
WHERE User_ID = '1' AND RegInc_Date = '2017-12-14';
COMMIT TRANSACTION;
```

Before UPDATE

Results Messages

	User_ID	User_Login	User_Pswrd	User_Email	First_Name	Last_Name	User_Age	User_Status	Reg_Date
1	1	alxbro69	Pfd234%4	alexbroman@gmail.com	Alex	Broman	48	Married	2017-11-13 22:55:19.443
2	2	kardug15	P@$sw99	karan.dugry@yahoo.com	Karan	Dugry	NULL	Single	2017-11-13 22:55:19.443
3	3	jabrone	forgotten	jamesbrone@private.com	James	Brone	35	Married	2017-11-13 22:55:19.443

	Balance_ID	User_ID	Pos_Balance	Neg_Balance	Balance_Date
1	1	2	1025.40	873.90	2017-11-14 00:15:32.313
2	2	3	768.37	1352.19	2017-11-14 00:15:32.313
3	3	1	532.94	921.15	2017-11-14 00:15:32.313

	RI_ID	User_ID	Income_ID	Description	RegInc_Amount	RegInc_Date	RegInc_Frequency	Active
1	1	3	1	My salary	1890.50	2017-11-19	14	1
2	2	2	1	Biweekly	1265.35	2017-11-28	14	1
3	3	1	2	Montly c...	580.17	2017-12-14	28	1

After UPDATE

Results Messages

	User_ID	User_Login	User_Pswrd	User_Email	First_Name	Last_Name	User_Age	User_Status	Reg_Date
1	1	alxbro69	Pfd234%4	alexbroman@gmail.com	Alex	Broman	48	Married	2017-11-13 22:55:19.443
2	2	kardug15	P@$sw99	karan.dugry@yahoo.com	Karan	Dugry	NULL	Single	2017-11-13 22:55:19.443
3	3	jabrone	forgotten	jamesbrone@private.com	James	Brone	35	Single	2017-11-13 22:55:19.443

	Balance_ID	User_ID	Pos_Balance	Neg_Balance	Balance_Date
1	1	2	1025.40	873.90	2017-11-14 00:15:32.313
2	2	3	1390.20	1352.19	2017-11-14 21:15:35.100
3	3	1	532.94	921.15	2017-11-14 00:15:32.313

	RI_ID	User_ID	Income_ID	Description	RegInc_Amount	RegInc_Date	RegInc_Frequency	Active
1	1	3	1	My salary	1890.50	2017-11-19	14	1
2	2	2	1	Biweekly	1265.35	2017-11-28	14	1
3	3	1	2	Montly commission	450.00	2017-12-14	28	1

```
BEGIN TRANSACTION;
DELETE FROM Users
WHERE User_ID = '3';

DELETE FROM Expense
WHERE Expense_ID = '2';

DELETE FROM Income
WHERE Income_ID = '2';
ROLLBACK TRANSACTION;
```

The tables are being deleted were chosen for a certain reason. As you may recall, on the beginning when we created all the tables we also created primary and foreign keys for them (by doing this we set up relationships between tables) and determined actions for our constraints. And now it is a good opportunity to check how our settings work.

When we delete user with ID 3, every record related to this user gets deleted as well. If we delete primary keys from Income and Expense tables, other tables that contain foreign keys referencing to them will set up NULL value in appropriate rows. After rollback all the data is back.

Before DELETE

Results Messages

	User_ID	User_Login	User_Pswrd	User_Email	First_Name	Last_Name	User_Age	User_Status	Reg_Date
1	1	alxbro69	Pfd234%4	alexbroman@gmail.com	Alex	Broman	48	Married	2017-11-13 22:55:19.443
2	2	kardug15	P@$sw99	karan.dugry@yahoo.com	Karan	Dugry	NULL	Single	2017-11-13 22:55:19.443
3	3	jabrone	forgotten	jamesbrone@private.com	James	Brone	35	Single	2017-11-13 22:55:19.443

	Expense_ID	Exp_Type
1	1	Mortgage
2	2	Credit card
3	3	Grocery

	Income_ID	Inc_Type
1	1	Salary
2	2	Commission
3	3	Bonus

	Balance_ID	User_ID	Pos_Balance	Neg_Balance	Balance_Date
1	1	2	1025.40	873.90	2017-11-14 00:15:32.313
2	2	3	1390.20	1352.19	2017-11-14 21:15:35.100
3	3	1	532.94	921.15	2017-11-14 00:15:32.313

	RI_ID	User_ID	Income_ID	Description	RegInc_Amount	RegInc_Date	RegInc_Frequency	Active
1	1	3	1	My salary	1890.50	2017-11-19	14	1
2	2	2	1	Biweekly	1265.35	2017-11-28	14	1
3	3	1	2	Montly c...	450.00	2017-12-14	28	1

	CI_ID	User_ID	Income_ID	CasInc_Amount	CasInc_Date
1	1	2	3	370.54	2017-11-14 19:31:05.737
2	2	1	2	780.60	2017-11-14 19:31:05.737
3	3	3	3	500.00	2017-11-14 19:31:05.737

	RE_ID	User_ID	Expense_ID	Description	Limit	RegExp_Amount	Statement	RegExp_Date	RegExp_Frequency	Paid_From	Active
1	1	1	1	First house	NULL	520.00	NULL	2017-11-17 19:50:04.633	14	Positive balance	1
2	2	2	2	RBC bank	2000.00	837.96	420.47	2017-11-20 19:50:04.633	28	Positive balance	1
3	3	3	2	Scotia ba...	1500.00	796.30	520.50	2017-11-24 19:50:04.633	28	Positive balance	1

	CE_ID	User_ID	Expense_ID	CasExp_Amount	CasExp_Date	Paid_From
1	1	1	3	320.95	2017-11-14 20:02:02.133	Positive balance
2	2	2	3	495.40	2017-11-14 20:02:02.133	Credit card RBC bank
3	3	3	3	178.26	2017-11-14 20:02:02.133	Credit card Scotia b...

After DELETE

Results Messages

	User_ID	User_Login	User_Pswrd	User_Email	First_Name	Last_Name	User_Age	User_Status	Reg_Date
1	1	alxbro69	Pfd234%4	alexbroman@gmail.com	Alex	Broman	48	Married	2017-11-13 22:55:19.443
2	2	kardug15	P@$sw99	karan.dugry@yahoo.com	Karan	Dugry	NULL	Single	2017-11-13 22:55:19.443

	Expense_ID	Exp_Type
1	1	Mortgage
2	3	Grocery

	Income_ID	Inc_Type
1	1	Salary
2	3	Bonus

	Balance_ID	User_ID	Pos_Balance	Neg_Balance	Balance_Date
1	1	2	1025.40	873.90	2017-11-14 00:15:32.313
2	3	1	532.94	921.15	2017-11-14 00:15:32.313

	RI_ID	User_ID	Income_ID	Description	RegInc_Amount	RegInc_Date	RegInc_Frequency	Active
1	2	2	1	Biweekly	1265.35	2017-11-28	14	1
2	3	1	NULL	Montly commission	450.00	2017-12-14	28	1

	CI_ID	User_ID	Income_ID	CasInc_Amount	CasInc_Date
1	1	2	3	370.54	2017-11-14 19:31:05.737
2	2	1	NULL	780.60	2017-11-14 19:31:05.737

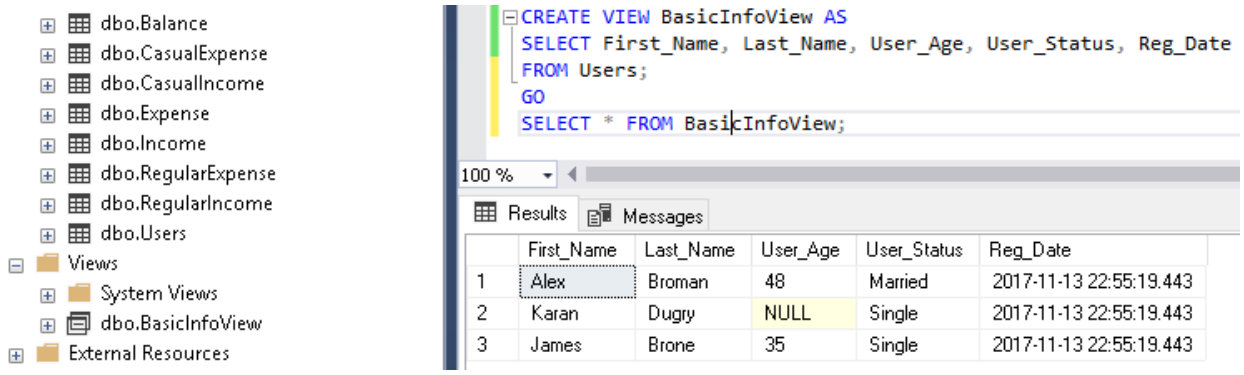
	RE_ID	User_ID	Expense_ID	Description	Limit	RegExp_Amount	Statement	RegExp_Date	RegExp_Frequency	Paid_From	Active
1	1	1	1	First house	NULL	520.00	NULL	2017-11-17 19:50:04.633	14	Positive balance	1
2	2	2	NULL	RBC bank	2000.00	837.96	420.47	2017-11-20 19:50:04.633	28	Positive balance	1

	CE_ID	User_ID	Expense_ID	CasExp_Amount	CasExp_Date	Paid_From
1	1	1	3	320.95	2017-11-14 20:02:02.133	Positive balance
2	2	2	3	495.40	2017-11-14 20:02:02.133	Credit card RBC bank

Data access restriction with view

Let's say, one of the support representatives wants to see basic information about registered users. For him or her it's not necessary to have access to user's login and password. View in this case will hide part of the information:

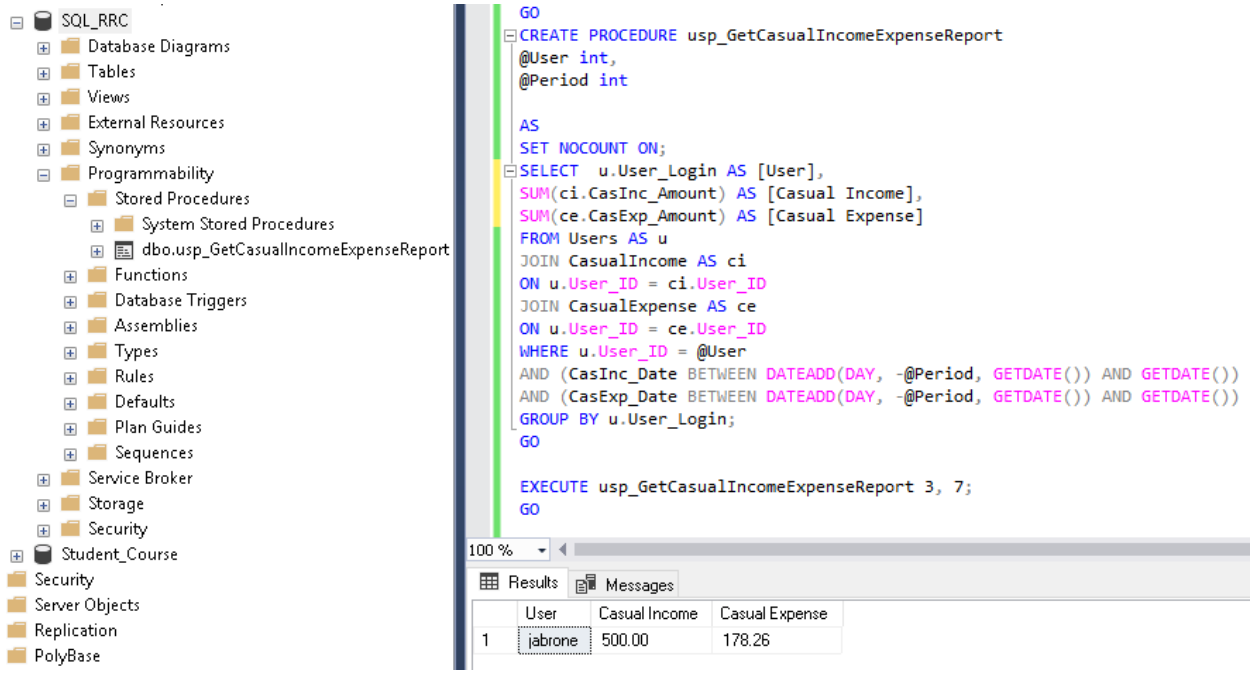
```
CREATE VIEW BasicInfoView AS
SELECT First_Name, Last_Name, User_Age, User_Status, Reg_Date
FROM Users;
GO
SELECT * FROM BasicInfoView;
```



Stored procedure

Working with money in most cases involves accounting and analysis. Your earnings and expenses have timestamp, for instance, during the last week each day you were spending different amount of money on your needs. Same applies to income – you could get casual income on some days during that week. And now you would like to know exactly how much you spent and earned in total. Stored procedure proposed below will give us the answer on such question: how much was earned and spent by certain user for the last N days:

```
CREATE PROCEDURE usp_GetCasualIncomeExpenseReport

@User int,
@Period int

AS
SET NOCOUNT ON;
SELECT  u.User_Login AS [User],
SUM(ci.CasInc_Amount) AS [Casual Income],
SUM(ce.CasExp_Amount) AS [Casual Expense]
FROM Users AS u
JOIN CasualIncome AS ci
ON u.User_ID = ci.User_ID
JOIN CasualExpense AS ce
ON u.User_ID = ce.User_ID
WHERE u.User_ID = @User
AND (CasInc_Date BETWEEN DATEADD(DAY, -@Period, GETDATE()) AND GETDATE())
AND (CasExp_Date BETWEEN DATEADD(DAY, -@Period, GETDATE()) AND GETDATE())
GROUP BY u.User_Login;
GO

EXECUTE usp_GetCasualIncomeExpenseReport 3, 7;
GO
```

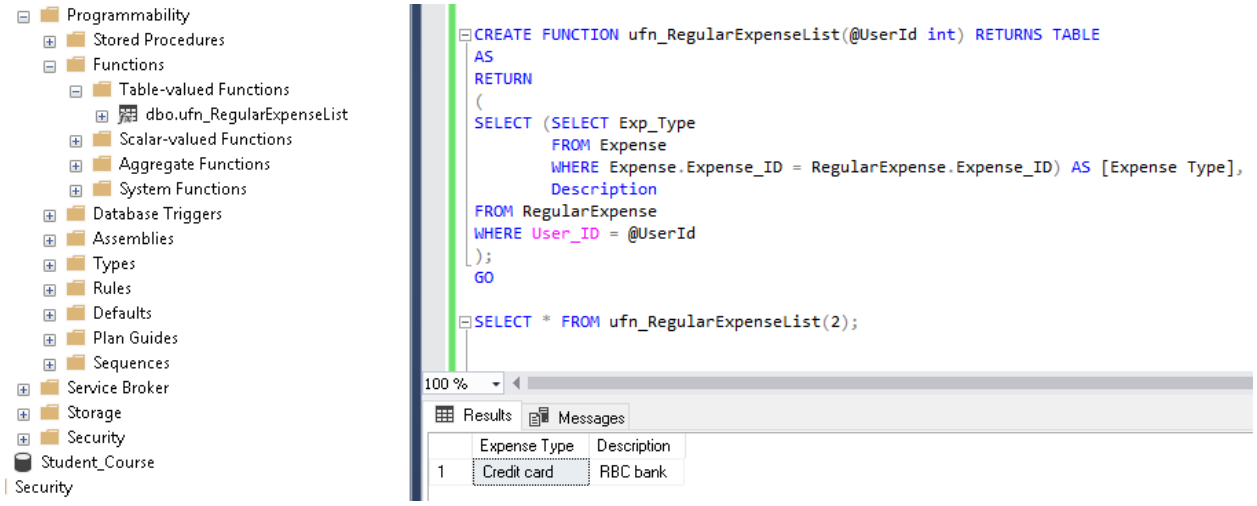


As you may noticed, within stored procedure were utilized JOIN clause to get columns from multiple tables and aggregate function SUM( ) to summarize all the values in two columns that meet search requirement.

User defined function

It could be useful at some point to know all the regular expenses for certain user in the database. Function below allows us to compile complete list of all user's regular expenses:

```
CREATE FUNCTION ufn_RegularExpenseList(@UserId int) RETURNS TABLE
AS
RETURN
(
SELECT (SELECT Exp_Type
        FROM Expense
        WHERE Expense.Expense_ID = RegularExpense.Expense_ID) AS [Expense Type],
        Description
FROM RegularExpense
WHERE User_ID = @UserId
);
GO

SELECT * FROM ufn_RegularExpenseList(2);
```

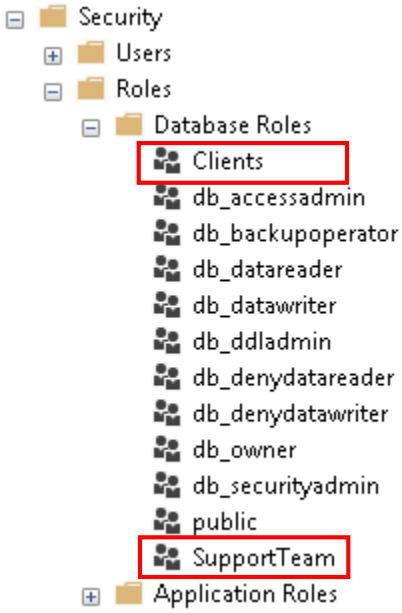


Current user defined function uses sub query inside to retrieve values from Exp_Type column from table Expense which match to values within Expense_ID column of the RegularExpense table.

Database access control

Let's create 2 types of roles and assign one user to each role.

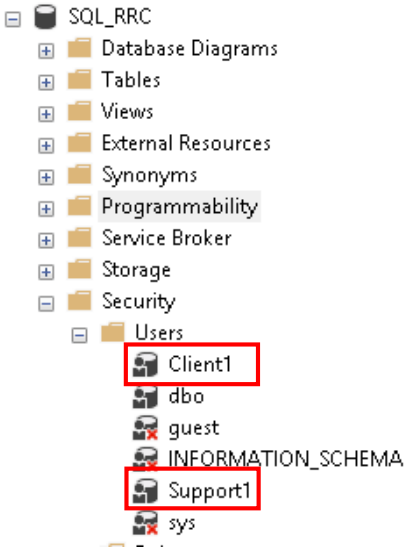
First we create 2 new roles: Clients and SupportTeam




```
--Database access control part

CREATE ROLE SupportTeam;
CREATE ROLE Clients;
```

```
100 %
Messages
Commands completed successfully.
```

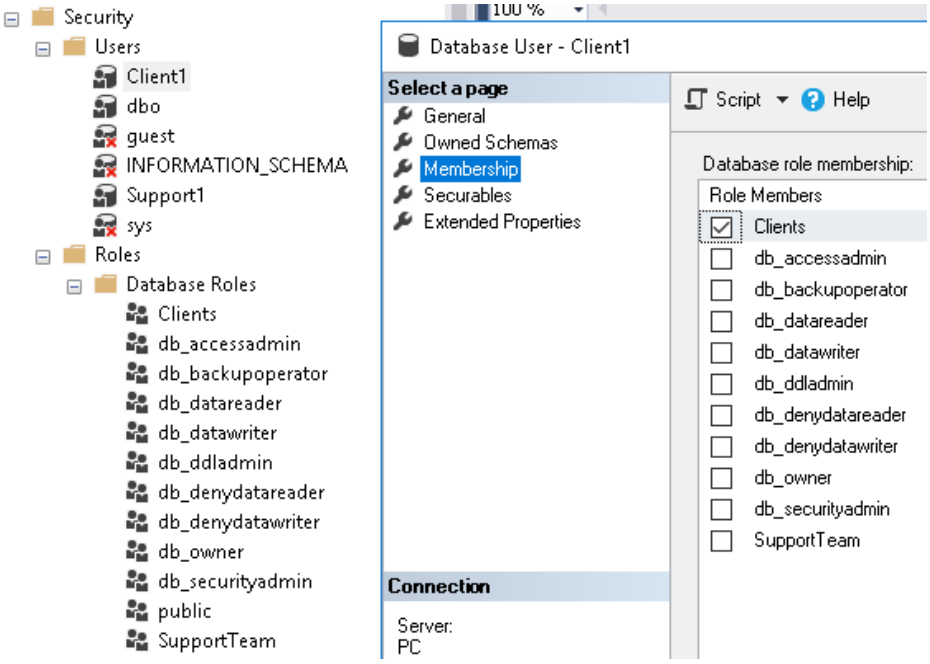
Then we create 2 new users with passwords assigned and add 1 user to each role



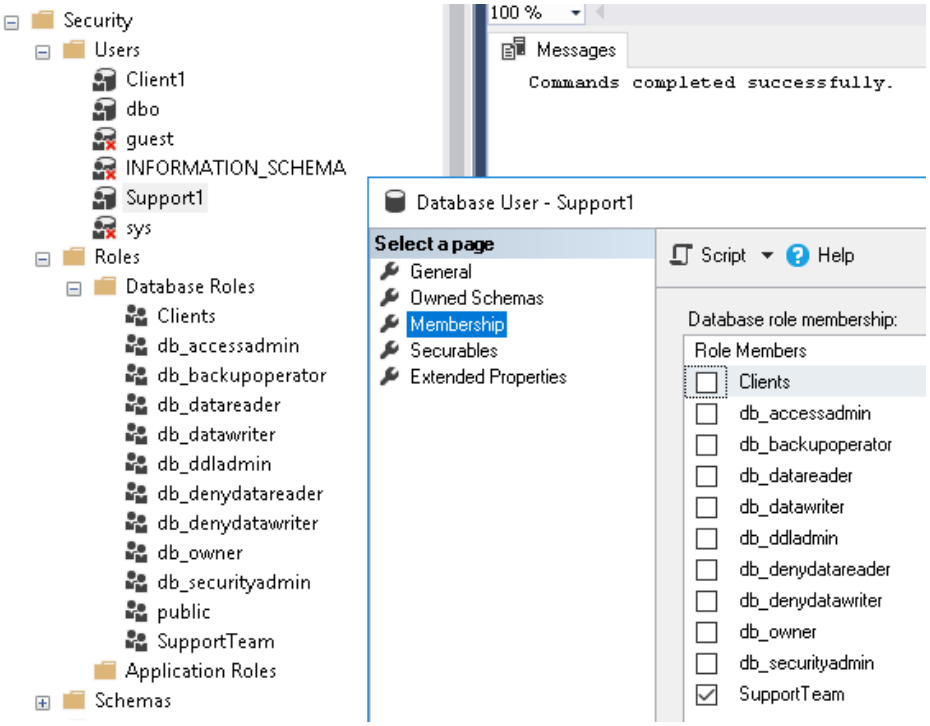

```
CREATE USER Support1 WITH PASSWORD = 'dgEFG@128';
GO
CREATE USER Client1 WITH PASSWORD = 'p@$Swd724';
GO

ALTER ROLE SupportTeam ADD MEMBER Support1;
GO
ALTER ROLE Clients ADD MEMBER Client1;
GO
```

```
100 %
Messages
Commands completed successfully.
```

User Client1 was assigned to Clients role



User Support1 was assigned to SuportTeam role

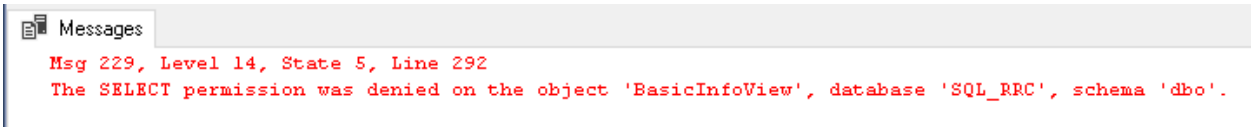


Earlier the view and stored procedure were created. We will grant permission to use SELECT statement on view BasicInfoView to the SupportTeam role and permission to use EXECUTE statement on stored procedure usp_GetCasualIncomeExpenseReport to the Clients role:

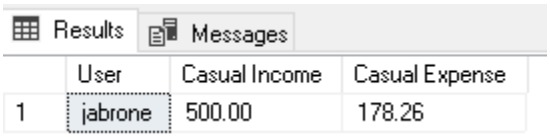
```
GRANT SELECT ON BasicInfoView TO SupportTeam;
GO

GRANT EXECUTE ON usp_GetCasualIncomeExpenseReport TO Clients;
GO
```

Let's look at a few examples that demonstrate permissions in action. We'll start by running an EXECUTE AS statement to change the execution context of our session to the Client1 and Support1 database user accounts in our database. For each user we will perform SELECT statement for the view and EXECUTE statement for the stored procedure:

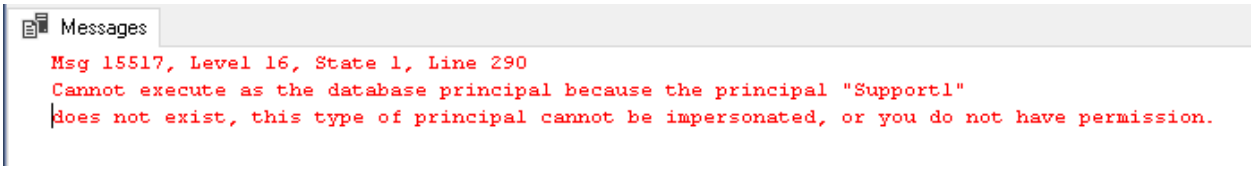
```
EXECUTE AS USER = 'Client1'

SELECT * FROM BasicInfoView;
```



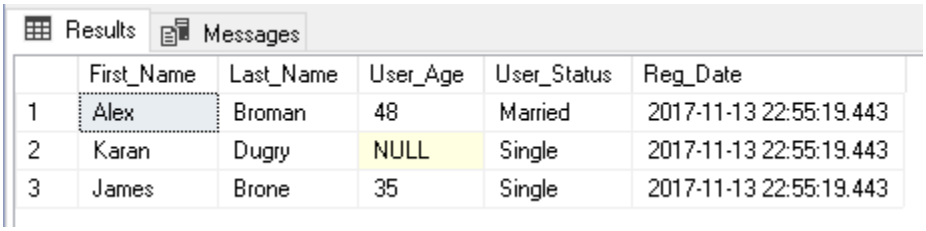

```
EXECUTE usp_GetCasualIncomeExpenseReport 3, 7;
GO
```



Now we will login under Support1. Before we can do that, however, we must run a REVERT statement to return to the context of the user that does have the necessary permissions, otherwise we can get error message:



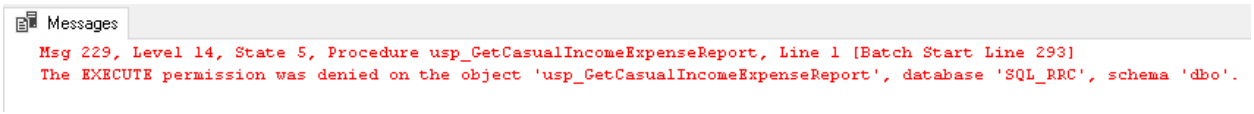
```
EXECUTE AS USER = 'Support1'

SELECT * FROM BasicInfoView;
```



Results | Messages

	First_Name	Last_Name	User_Age	User_Status	Reg_Date
1	Alex	Broman	48	Married	2017-11-13 22:55:19.443
2	Karan	Dugry	NULL	Single	2017-11-13 22:55:19.443
3	James	Brone	35	Single	2017-11-13 22:55:19.443

```
EXECUTE usp_GetCasualIncomeExpenseReport 3, 7;
GO
```



Messages

```
Msg 229, Level 14, State 5, Procedure usp_GetCasualIncomeExpenseReport, Line 1 [Batch Start Line 293]
The EXECUTE permission was denied on the object 'usp_GetCasualIncomeExpenseReport', database 'SQL_RRC', schema 'dbo'.
```